	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorio de Computacion Salas A y B

Profesor(a): Rene Adrian Davila Perez

Asignatura: Programacion orientada a objetos

Grupo: 01

No de practica(s): 4

Integrante(s): 322001549, 322006829, 322125533, 425131152, 322103351

No de lista o brigada: 07

Semestre: 2026-1

Fecha de entrega: 19 de septiembre del 2025

Observaciones:

Calificacion:

Índice

1. Introducción

1.1. Planteamiento del Problema

Se hace una descripción del problema a resolver Se requiere que se cree un programa que calcule la distancia entre dos puntos dados por el usuario. Además, se necesita que todo sea con ayuda de la biblioteca **Java Swing**, puesto que se debe desplegar una pequeña pantalla que te muestre los puntos que ingresaste y justo abajo un botón que diga "*Calcular*". Una vez se presione el botón se tiene que desplegar otra pantalla emergente que muestre la distancia calculada.

1.2. Objetivos

- Conocer las funciones de la biblioteca **Java Swing**
- Analizar la sintaxis y un poco de la aplicación de la "*Herencia*"
- Practicar el uso e implementación de los constructores

2. Marco teórico

2.1. POO

La programación orientada a objetos es un modelo de programación en el que el diseño de software se organiza alrededor de datos u objetos, en vez de usar funciones y lógica. Se enfoca en los objetos que los programadores necesitan manipular, en lugar de centrarse en la lógica necesaria para esa manipulación. Un objeto se puede definir como un campo de datos con atributos y comportamientos únicos.[1]

Por tanto, la principal característica de este tipo de programación es que soporta objetos, que tienen un tipo o clase asociado. Esas clases pueden heredar atributos de una clase superior o superclase. Por esa razón, este enfoque de programación se utiliza en programas grandes y complejos que se deben actualizar con cierta regularidad.[1]

En Java, las clases y los objetos son bloques de construcción fundamentales de la programación orientada a objetos. Una clase es un plano para crear objetos, proporcionar valores iniciales a las variables miembro e implementar comportamientos (métodos). Un objeto es una instancia de una clase, que representa una entidad concreta con un estado y un comportamiento definidos.[2]

2.2. Clase

Una clase en Java define un nuevo tipo de datos que puede incluir campos (variables) y métodos para definir el comportamiento de los objetos creados a partir de la clase: [2]

```
class ClassName {
    // Fields
    dataType fieldName;

    // Constructor
    ClassName(parameters) {
        // Initialize fields
    }

    // Methods
    returnType methodName(parameters) {
        // Method body
    }
}
```

De donde donde:

- ClassName: El nombre de la clase.
- fieldName: Variables que contienen el estado de la clase.
- Constructor: Método especial utilizado para inicializar objetos.
- methodName: Funciones definidas dentro de la clase para realizar acciones.

2.3. Objeto

Un objeto es una instancia de una clase. Se crea utilizando la palabra clave new seguida del constructor de la clase: [2]

```
ClassName objectName = new ClassName(arguments);
```

De donde:

- objectName: El nombre del objeto.
- arguments: Valores pasados al constructor para inicializar el objeto

A continuación un ejemplo incluyendo ambos conceptos:

```java
class Car {
    // Fields
    String color;
    int year;

    // Constructor
    Car(String color, int year) {
        this.color = color;
        this.year = year;
    }

    // Method
    void displayInfo() {
        System.out.println("Car color: " + color + ", Year: " +
    year);
    }
}

public class Main {
    public static void main(String[] args) {
        // Creating an object
        Car myCar = new Car("Red", 2020);
        myCar.displayInfo(); // Output: Car color: Red, Year: 2020
    }
}
```

En este ejemplo, la clase Car tiene los campos color y year, un constructor para inicializar estos campos, y un método displayInfo() para mostrar los detalles del coche. Se crea un objeto myCar utilizando la clase Car.

2.4. Constructor

En Java, un constructor es un método especial utilizado para inicializar objetos. A diferencia de los métodos normales, los constructores se invocan cuando se crea una instancia de una clase. Tienen el mismo nombre que la clase y no tienen tipo de retorno. Los constructores son esenciales para establecer los valores iniciales de los atributos del objeto y prepararlo para su uso.[3] Tipos de constructores:

1. Constructor por defecto: El compilador Java proporciona automáticamente un constructor por defecto si no se definen constructores explícitamente en la clase. Inicializa el objeto con valores por defecto.
2. Constructor sin argumentos: Un constructor sin argumentos es definido explícitamente por el programador y no recibe ningún parámetro. Es similar al constructor por defecto, pero puede incluir código de inicialización personalizado.[3]
3. Constructor parametrizado: Un constructor parametrizado acepta argumentos para inicializar un objeto con valores específicos. Esto permite una inicialización más flexible y controlada de los atributos de los objetos.[3]
4. Copiar Constructor: Un constructor de copia se utiliza para crear un objeto nuevo como copia de un objeto existente. Java no proporciona un constructor de copia por defecto; sin embargo, puede implementarse manualmente.[3]

A continuación sus sintaxis de constructor por defecto:

```
class ClassName {
    // Constructor
    ClassName() {
        // Initialization code
    }

    // Parameterized Constructor
    ClassName(dataType parameter1, dataType parameter2) {
        // Initialization code using parameters
    }

    // Copy Constructor
    ClassName(ClassName obj) {
        // Initialization code to copy attributes
    }
}
```

Ejemplo de constructor:

```
public class Car {
    String model;
    int year;

    // Default Constructor
    public Car() {
        model = "Unknown";
        year = 0;
    }

    public static void main(String[] args) {
        Car car = new Car();
        System.out.println("Model: " + car.model + ", Year: " +
    car.year);
    }
}
```

En este ejemplo, la clase Car tiene un constructor por defecto que inicializa los atributos model y year con valores por defecto. Cuando se crea un objeto Car, se le asignan estos valores por defecto. [3]

Ejemplo de constructor parametrizado:

```
public class Car {
    String model;
    int year;

    // Parameterized Constructor
    public Car(String model, int year) {
        this.model = model;
        this.year = year;
    }

    public static void main(String[] args) {
        Car car = new Car("Toyota", 2021);
        System.out.println("Model: " + car.model + ", Year: " +
    car.year);
    }
}
```

Ejemplo de sobrecarga de constructores:

```
public class Car {
    String model;
    int year;

    // No-Argument Constructor
    public Car() {
        model = "Unknown";
        year = 0;
    }

    // Parameterized Constructor
    public Car(String model, int year) {
        this.model = model;
        this.year = year;
    }

    public static void main(String[] args) {
        Car car1 = new Car();
        Car car2 = new Car("Honda", 2022);
        System.out.println("Car1 -> Model: " + car1.model + ",
    Year: " + car1.year);
        System.out.println("Car2 -> Model: " + car2.model + ",
    Year: " + car2.year);
    }
}
```

Este ejemplo demuestra la sobrecarga de constructores en la clase Car, donde se definen tanto un constructor sin argumentos como un constructor parametrizado. Esto permite crear objetos con valores predeterminados o específicos.[3]

2.5. Swing

Con la evolución de la tecnología y el surgimiento del concepto de multiplataforma con la versión 1.2 de Java (anteriormente llamado Playground) o Java 2, en el año 1998 surge la API gráfica de Swing Java Swing que integra la JFC (Java Foundation Classes) las cuáles reúnen componentes para la construcción de una GUI (Graphical User Interface - Interface Gráfica de Usuario).[4]

La biblioteca posibilita el desarrollo de interfaces elaboradas para un ambiente de computación heterogéneo con interacción más agradable. Las aplicaciones pueden seguir una apariencia y comportamiento de la plataforma nativa de Java o alguna plataforma personalizada. Permite el gerenciamiento de layouts, manejo de eventos, manipulación de imágenes en dos dimensiones - 2D y abarca diversos idiomas, entre otras.[4]

Además de esto, extiende el beneficio de computación con Java y la API de accesibilidad que funciona con dispositivos de entradas y salidas, como lectores de pantalla, terminales en Braille y otros. Sus componentes son independientes de la plataforma con el patrón MCV (Modelob-Vista-Controlador).[4]

En la programación orientada a objetos los componentes son Clases/Objetos que implementan la interfaz gráfica de usuario. Los componentes básicos proporcionados por la biblioteca Java Swing son:

- JButton
- JCheckBoxName
- JLabelGenericName
- JTextAreaName

- JTextFieldName
- JPasswordField
- JRadioButtonName
- JOptionPane

2.5.1. Etiqueta (JLabel)

Estos componentes puede ser usado para mostrar textos simples, imágenes con textos. También pueden ser usados para mostrar los resultados de un proceso. Los métodos más usados son: [4]

- setText() - Permite colocar un valor de texto, puede ser usado para mostrar un resultado en este componente.
- setFont() - Permite definir el tipo y el estilo de la fuente (negrita y/o cursiva) y el tamaño de la fuente.
- tFont() - Permite definir el tipo y el estilo de la fuente (negrita y/o cursiva) y el tamaño de la fuente. setIcon() - Permite que un ícono o una imagen sea usada y mostrada en la pantalla

2.5.2. Cajas de Texto (JTextField, JPasswordField y JTextArea)

Estos componentes son usados para que el usuario introduzca un valor textural y capturarlo. JTextField permite introducir una línea de texto, JPasswordField permite introducir un valor pero lo muestra camuflado para no poder ver el valor introducido y JTextArea permite introducir varias líneas de texto. Los métodos más usados son:[4]

- setText() - Permite colocar un valor de texto, puede ser usado para mostrar un resultado en este componente.
- getText() - Permite recuperar el valor de una caja de texto.
- setFont() - Permite definir el tipo y el estilo de la fuente (negrita y/o cursiva) y el tamaño de la fuente.
- setEnabled() - Permite que la escritura sea deshabilitada y puede ser usado para mostrar un resultado como en un JLabel.

2.5.3. Botones (JRadioButton, JCheckBox y JButton)

Estos componentes permiten varias formas gráficas con las cuales el usuario puede interactuar, el JRadioButton permite elegir una opción dentro de un grupo de elecciones, en cuanto al JCheckBox permite seleccionar más de una de estas opciones. Finalmente los JButton son usados para ejecutar un evento cuando son presionados. [4]

2.5.4. Listas Desplegables (JComboBox)

Parecido al JRadioButton, permite seleccionar una opción de un grupo de alternativas pero sin ocupar espacio en la pantalla.

2.5.5. Cajas de Diálogo (JOptionPane)

Este permite mostrar una caja con un mensaje, un icono y botones. Un código de ejemplo sería: [4]

```
JOptionPane.showMessageDialog(null,     Mensaje     ,      Ttulo     ,
    JOptionPane.WARNING_MESSAGE);
```

El segundo y tercer parámetro indican el mensaje y el título, estas cajas pueden ser usadas para mostrar mensajes de confirmación, errores, avisos entre otras posibilidades.

2.5.6. Jerarquía de Componentes

Ahora que ya vimos sobre los componentes, podemos hablar sobre la jerarquía existente entre ellos dentro de la biblioteca Swing. Este rango es utilizado en la funcionalidad de estos elementos para la comprensión de una aplicación Swing y es comúnmente visualizada como un árbol de componentes. [4]

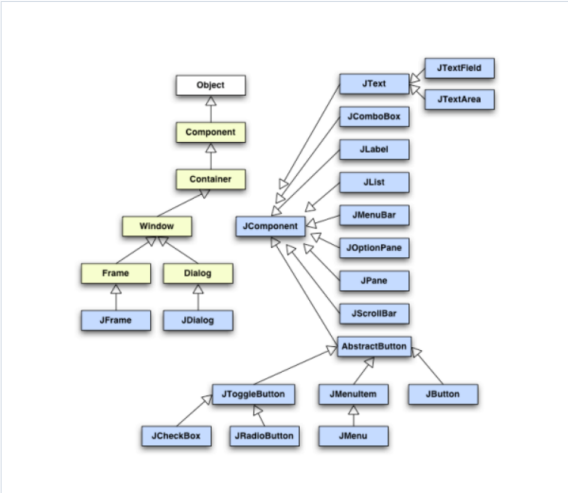


Figura 1: Jerarquia de componentes

Este árbol trabaja con tres tipos de elementos en una aplicación:

1. Contenedor de nivel superior: Es un elemento que generalmente se usa como base, es decir, proporcionar un lugar para usar otros elementos. Ejemplos de este tipo serían: JFrame, JDialog y JApplet.
2. Componente de nivel intermedio: Es un elemento que se usa solo para manejar la ubicación de los elementos de botones y etiquetas. Ejemplos de este tipo serían: JPanel, JScrollPane y JTabbedPane.
3. Contenedor Atómico: Es un elemento que no se usa para almacenar otros elementos, es una entidad autosuficiente, que sirve para presentar información al usuario o para recibir información proporcionada por el usuario. Algunos de los ejemplos de este tipo son: JButton, JLabel, JComboBox, JTextField y JTable.

Considerando todo lo anterior, se muestra a continuación un ejemplo:

```java
import java.awt.GridBagLayout;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextArea;
public class Main {
  public static void main(String[] args){
    // componente JFrame
   JFrame miJFrame = new JFrame("Ejemplo - Java Swing");
    miJFrame.setSize(500,300);
    // componente JPanel‘
    ‘JPanel miJPanel = new JPanel();‘
    ‘miJPanel.setSize(300, 300);‘
    // usamos este dise o para centrar los componentes de JPanel‘
    miJPanel.setLayout(new GridBagLayout());‘
    // componente JTextField‘
    JLabel miJLabel = new JLabel();
    miJLabel.setText("Dime tu opinion acerca de Java Swing:  ");
    // componente JTextArea
    JTextArea miJTextArea = new JTextArea(5,20);
    // conecta los componentes JLabel y JTextField en JPanel‘
    miJPanel.add(miJLabel);
    miJPanel.add(miJTextArea);
    // conectar Jpanel a JFrame‘
    miJFrame.add(miJPanel);
    // hacer visible JFrame
    miJFrame.setVisible(true);
  }
}
```

De modo que el resultado es el siguiente:

Figura 2: Resulado del codigo

3. Desarrollo

La práctica tiene como finalidad reforzar el uso de clases, objetos y métodos en Java, aplicando el concepto de orientación a objetos para calcular la distancia entre dos puntos en un plano cartesiano. Además, se buscó integrar el uso de interfaces gráficas (Swing) para mejorar la interacción con el usuario.

3.1. Descripción de la Implementación

El proyecto se estructuró en cuatro clases principales, cada una con una responsabilidad específica:

Algoritmo 1: Clase Puntos

Data: Coordenadas x y y
Result: Representación de un punto en el plano
Constructor por defecto inicializa $(1, 1)$;
Constructor con parámetros asigna valores a (x, y);
Método `toString()` devuelve la cadena:
"Punto (x = valor , y = valor)";

Algoritmo 2: Clase Mensajes

Result: Mensajes para mostrar al usuario
Método `mensaje()` devuelve el título de la práctica;
Método `mensaje(p1,p2,distancia)` devuelve cadena con:
P1, P2 y la distancia calculada;

Algoritmo 3: Clase Ventana

Data: Datos ingresados por el usuario
Result: Ventana gráfica que muestra la distancia
Crear ventana con un botón;
Cuando el botón es presionado:
Crear punto $p1$ con $(x1, y1)$;
Crear punto $p2$ con $(x2, y2)$;
Calcular distancia con fórmula:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Mostrar mensaje con resultados en un cuadro de diálogo;

Algoritmo 4: Clase Practica4

Data: Parámetros de entrada desde consola
Result: Inicialización del programa
Leer argumentos recibidos como `x=valor`;
Separar valores y guardarlos en un arreglo;
Crear objeto `Mensajes`;
Crear objeto `Ventana` con los datos;
Mostrar la ventana;

3.2. Pruebas Realizadas

El programa se ejecutó con diferentes parámetros desde la consola. Al presionar el botón en la ventana, se mostró un cuadro de diálogo con la información de los puntos y la distancia calculada.

Ejemplo 1

```
java Practica4 x1=2 y1=3 x2=6 y2=7
```

Resultado:

```
Punto (x = 2 , y = 3)
Punto (x = 6 , y = 7)
Distancia = 5.656854249492381
```

Ejemplo 2

Puntos $(0, 0)$ y $(3, 4)$:

```
Distancia = 5.0
```

Ejemplo 3

Puntos $(1, 1)$ y $(1, 1)$:

```
Distancia = 0.0
```

4. Resultados

- **Clase Puntos**

  1. Define dos atributos enteros (`x` y `y`) que representan las coordenadas de un punto.
  2. Permite inicializar un punto en el origen (constructor vacío) o en cualquier posición específica (constructor con parámetros).
  3. Implementa el método `toString()`, que convierte el objeto punto en una cadena de texto legible, lo que facilita mostrarlo en pantalla.

  En términos abstractos, esta clase se encarga de **modelar un objeto geométrico** y encapsular sus coordenadas.

```java
public class Puntos {
    int x = 1, y = 1;


    public Puntos() {}

    public Puntos(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public String toString() {
        return "Punto (x = " + x + " , y = " + y + ")";
    }
}

```

- **Clase Mensajes**

  1. Contiene un método que devuelve un mensaje de bienvenida estático.
  2. Incluye un método sobrecargado que recibe dos objetos `Puntos` y un valor de distancia.
  3. Construye una cadena de texto que une la información de los puntos y la distancia calculada para que el usuario la vea de forma clara.

  En otras palabras, esta clase abstrae el **formato de salida** y centraliza los mensajes mostrados al usuario.

```java
public class Mensajes {
    public String mensaje() {

        return "Practica 4 Equipo 7!";
    }
     public String mensaje(Puntos p1, Puntos p2, double distancia) {
        return p1 + "\n" +p2 + "\n" +
                "Distancia = " + distancia;
    }
}

```

- **Clase Ventana**

  1. Hereda de `JFrame`, lo que le permite generar una ventana gráfica.
  2. Define un botón que, al ser presionado, ejecuta una acción programada con un `ActionListener`.
  3. Dentro de la acción:

     a) Se crean dos objetos `Puntos` a partir de los valores que provienen de los argumentos de entrada.

     b) Se aplica la fórmula de la distancia euclidiana:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

     c) Se invoca al método de la clase `Mensajes` para generar el texto final.

     d) Se muestra un cuadro emergente (`JOptionPane`) con la información de los puntos y la distancia.

  Así, esta clase combina la **interfaz gráfica con el cálculo matemático y la visualización de resultados**.

```java
import javax.swing.*;
import java.awt.event.*;

public class Ventana extends JFrame {
    JButton boton;
    Mensajes controlador;
    Double[] datos;

    public Ventana(Mensajes controlador, Double[] datos) {
        this.controlador = controlador;
        this.datos = datos;

        setTitle(title:"Ejemplo");
        setSize(width:300, height:200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(c:null);

        boton = new JButton(text:"Haz clic aqui");
        boton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                Puntos p1 = new Puntos(datos[0].intValue(), datos[1].intValue());
                Puntos p2 = new Puntos(datos[2].intValue(), datos[3].intValue());
                double distancia = Math.sqrt(
                    Math.pow(p2.x - p1.x, b:2) + Math.pow(p2.y - p1.y, b:2)
                );

                // Usar el método mensaje con parámetros
                String mensaje = controlador.mensaje(p1, p2, distancia);

                JOptionPane.showMessageDialog(parentComponent:null, mensaje);
            }
        });

        add(boton);
    }
}

```

- **Clase Practica4**

  1. Es la clase principal con el método `main`.
  2. Toma los parámetros de entrada que llegan en el formato `x=10`, `y=20`, etc.
  3. Divide cada parámetro usando el símbolo -τ convierte el valor en un número de tipo `Double`.
  4. Almacena todos los valores en un arreglo de tipo `Double[]` para después ser utilizado.
  5. Crea un objeto de tipo `Ventana`, pasando como argumentos el controlador de mensajes y el arreglo con los datos.
  6. Finalmente, hace visible la ventana, dando inicio a la ejecución del programa gráfico.

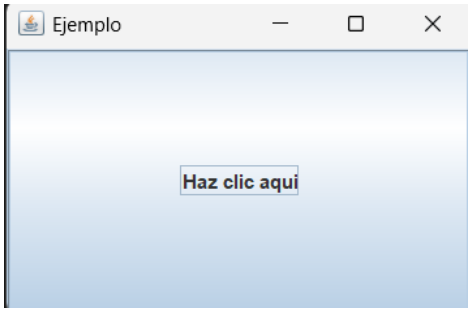
  En resumen, esta clase es la **puerta de entrada del programa**, encargada de preparar los datos y lanzar la interfaz gráfica.

```java
public class Practica4 {
    Run | Debug
    public static void main(String[] args) {
        String tijera;
        Double [] datos=new Double[4];
        for(int i=0;i<args.length;i++){
            tijera=args[i];//x=10
            String []Guardar=tijera.split(regex:"=");
            datos[i]=Double.parseDouble(Guardar[1]);
        }

        Mensajes controlador = new Mensajes();

        Ventana ventana = new Ventana(controlador,datos);
        ventana.setVisible(b:true);
    }

}

```

Primero para ejecutar tenemos que compilar los archivos .java y, debido a que son varios códigos, compilamos con javac *.java para compilar todos los archivos .java que estén dentro de la carpeta. Posteriormente al momento de ejecutar ya le tenemos que dar los parámetros que queremos calcular su distancia como por ejemplo en este casa "java Practica4 x1=2 y1=3 x2=6 y2=7".

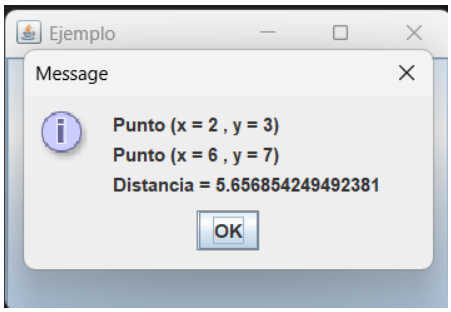
```
PS C:\Users\52551\Programacion\PracticasLab-4> cd Practica4
PS C:\Users\52551\Programacion\PracticasLab-4\Practica4> javac *.java
PS C:\Users\52551\Programacion\PracticasLab-4\Practica4> java Practica4 x1=2 y1=3 x2=6 y2=7
```

Al ejecutar el main (en este caso Practica4) se nos abre una ventana con un botón en el centro que nos llevara a otra ventana que nos desplegara los datos.



Despues de presionar el "Haz clic aqui", nos desplegara la ventana con los parametros y sus resultados osea la ubicacion de los puntos y la distancia entre ellos utilizando la formula de:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$



5. Conclusiones

Se logró cumplir de manera satisfactoria cada uno de los objetivos planteados en la práctica. Durante el proceso, no solo fue posible desplegar pantallas emergentes, sino también explorar y comprender mejor el funcionamiento de diversas herramientas que ofrece Java Swing, tales como JButton, JLabel, JTextField y JOptionPane, las cuales resultaron fundamentales para la interacción con el usuario.

Además, se adquirió un conocimiento más sólido sobre la sintaxis de herencia en Java, haciendo uso de la palabra reservada extends, lo que permitió comprender cómo una clase puede heredar atributos y métodos de otra para optimizar el código y favorecer la reutilización. Asimismo, se implementaron de manera satisfactoria constructores, los cuales desempeñaron un papel esencial en la creación y despliegue de las diferentes pantallas emergentes que conformaron el proyecto.

Referencias

[1] U. Europea. (2022) Programación orientada a objetos: qué es, características y ejemplos. Accedido: 16 de septiembre de 2025. [Online]. Available: https://universidadeuropea.com/blog/programacion-orientada-objetos/

[2] DataCamp. (2025) Clases y objetos en java. Accedido: 16 de septiembre de 2025. [Online]. Available: https://www.datacamp.com/es/doc/java/classes-and-objects

[3] ——. (2025) Java constructors. Accedido: 16 de septiembre de 2025. [Online]. Available: https://www.datacamp.com/es/doc/java/constructors

[4] L. E. Puig. (2022) ¿qué es la biblioteca swing? Publicado: 31 de julio de 2022; Accedido: 16 de septiembre de 2025. [Online]. Available: https://www.aluracursos.com/blog/biblioteca-swing